ELEC 677 – Fall 2016
Assignment 1

Due: Tuesday Oct 4, 11AM via GradeScope: PDF report + link to code

1 Backpropagation in a Simple Neural Network

In this problem, you will learn how to implement the backpropagation algorithm for a simple neural network. To make your job easier, we provide you with starter code in `three_layer_neural_network.py`. You will fill in this starter code to build a 3-layer neural network (see Fig. 1) and train it using backpropagation.

a) Dataset

We will use the `Make-Moons` dataset available in Scikit-learn. Data points in this dataset form two interleaving half circles corresponding to two classes (e.g. "female" and "male"). In the main() function of `three_layer_neural_network.py`, uncomment the "generate and visualize Make-Moons dataset" section (see below) and run the code. Include the generated figure in your report.

```
# generate and visualize Make-Moons dataset
X, y = generate_data()
plt.scatter(X[:, 0], X[:, 1], s=40, c=y, cmap=plt.cm.Spectral)
```

b) Activation Function

Tanh, Sigmoid and ReLU are popular activation functions used in neural networks. You will implement them and their derivatives.

1. Implement function `actFun(self, z, type)` in `three_layer_neural_network.py`. This function computes the activation function where `z` is the net input and `type` $\in$ {'Tanh', 'Sigmoid', 'ReLU'}.

2. Derive the derivatives of Tanh, Sigmoid and ReLU

3. Implement function `diff_actFun(self, z, type)` in `three_layer_neural_network.py`. This function computes the derivatives of Tanh, Sigmoid and ReLU.

c) Build the Neural Network

Lets now build a 3-layer neural network of one input layer, one hidden layer, and one output layer. The number of nodes in the input layer is determined by the dimensionality of our data, 2. The number of nodes in the output layer is determined by the number of classes we have, also 2. The input to the network will be x- and y- coordinates and its output will be two probabilities, one for class 0 ("female") and one for class 1 ("male"). The network looks like the following.

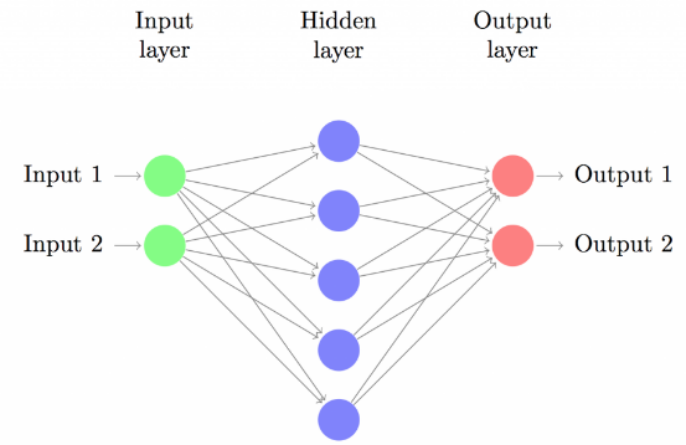

Figure 1: A three-layer neural network

Mathematically, the network is defined as follows.

$$z_1 = W_1 x + b_1 \tag{1}$$
$$a_1 = \text{actFun}(z_1) \tag{2}$$
$$z_2 = W_2 a_1 + b_2 \tag{3}$$
$$a_2 = \hat{y} = \text{softmax}(z_2) \tag{4}$$

where z_i is the input of layer i and a_i is the output of layer i after applying the activation function. $\theta \equiv \{W_1, b_1, W_2, b_2\}$ are the parameters of this network, which we need to learn from the training data.

If we have N training examples and C classes then the loss for the prediction $\hat{y}$ with respect to the true labels y is given by:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{n \in N} \sum_{i \in C} y_{n,i} \log \hat{y}_{n,i} \tag{5}$$

Note that y are one-hot-encoding vectors and $\hat{y}$ are vectors of probabilities.

1. In three_layer_neural_network.py, implement the function `feedforward(self, X, actFun)`. This function builds a 3-layer neural network and computes the two probabilities (`self.probs` in the code or a_2 in Eq. 4), one for class 0 and one for class 1. `X` is the input data, and `actFun` is the activation function. You will pass the function `actFun` you implemented in part b into `feedforward(self, X, actFun)`.

2. In three_layer_neural_network.py, fill in the function `calculate_loss(self, X, y)`. This function computes the loss for prediction of the network. Here `X` is the input data, and `y` is the given labels.

d) Backward Pass - Backpropagation

It's time to implement backpropagation, finally!

1. Derive the following gradients: $\frac{\partial L}{\partial W_2}$, $\frac{\partial L}{\partial b_2}$, $\frac{\partial L}{\partial W_1}$, $\frac{\partial L}{\partial b_1}$ mathematically

2. In three_layer_neural_network.py, implement the function `backprop(self, X, y)`. Again, X is the input data, and y is the given labels. This function implements backpropagation (i.e., computing the gradients above).

e) Time to Have Fun - Training!

You already have all components needed to run the training. In three_layer_neural_network.py, we also provide you function `visualize_decision_boundary(self, X, y)` to visualize the decision boundary. Let's have fun with your network now.

1. Train the network using different activation functions (Tanh, Sigmoid and ReLU). Describe and explain the differences that you observe. Include the figures generated in your report. In order to train the network, uncomment the `main()` function in three_layer_neural_network.py, take out the following lines, and run three_layer_neural_network.py.

```
plt.scatter(X[:, 0], X[:, 1], s=40, c=y, cmap=plt.cm.Spectral)
plt.show()
```

2. Increase the number of hidden units (`nn_hidden_dim`) and retrain the network using Tanh as the activation function. Describe and explain the differences that you observe. Include the figures generated in your report.

f) Even More Fun - Training a Deeper Network!!!:

Let's have some more fun and be more creative now. Write your own `n_layer_neural_network.py` that builds and trains a neural network of n layers. Your code must be able to accept as parameters (1) the number of layers and (2) layer size. We provide you hints below to help you organize and implement the code, but if you have better ideas, please feel free to implement them and ignore our hints. In your report, please tell us why you made the choice(s) you did.

Hints:

1. Create a new class, e.g `DeepNeuralNetwork`, that inherits `NeuralNetwork` in `three_layer_neural_network.py`
2. In `DeepNeuralNetwork`, change function `feedforward`, `backprop`, `calculate_loss` and `fit_model`
3. Create a new class, e.g. `Layer()`, that implements the feedforward and backprop steps for a single layer in the network
4. Use `Layer.feedforward` to implement `DeepNeuralNetwork.feedforward`
5. Use `Layer.backprop` to implement `DeepNeuralNetwork.backprop`
6. Notice that we have L2 weight regularizations in the final loss function in addition to the cross entropy. Make sure you add those regularization terms in `DeepNeuralNetwork.calculate_loss` and their derivatives in `DeepNeuralNetwork.fit_model`.

Train your network on the `Make_Moons` dataset using different number of layers, different layer sizes, different activation functions and, in general, different network configurations. In your report, include generated images and describe what you observe and what you find interesting (e.g. decision boundary of deep vs shallow neural networks).

Next, train your network on another dataset different from `Make_Moons`. You can choose datasets provided by Scikit-learn (more details here) or any dataset of your interest. Make sure that you have the correct number of input and output nodes. Again, play with different network configurations. In your report, describe the dataset you choose and tell us what you find interesting.

Be curious and creative!!! You are exploring Deep Learning. :)

Submission Instructions

Every student must submit their work in PDF format, providing intermediate and final results as well as any necessary code. Submit your homework on Gradescope.

Collaboration Policy

Collaboration both inside and outside class is encouraged. You may talk to other students for general ideas and concepts, but individual write-ups must be done independently.

Plagiarism

Plagiarism of any form will not be tolerated. You are expected to credit all sources explicitly.